Konzeptionelle Modellierung von Informationssystemen

Ehrich, Hans-Dieter

Veröffentlicht in:
Jahrbuch 1997 der Braunschweigischen Wissenschaftlichen Gesellschaft, S.43-54

Verlag Erich Goltze KG, Göttingen

HANS-DIETER EHRICH

Konzeptionelle Modellierung von Informationssystemen*

Braunschweig, 10. Oktober 1997**

1 Einleitung

Wenn Sie eine Urlaubsreise buchen wollen, gehen Sie in ein Reisebüro. Dort wird Ihnen eine freundliche Angestellte am Bildschirm sagen können, ob zur gewünschten Zeit Flüge, Hotelzimmer und Mietwagen der gewünschten Kategorie verfügbar sind. Wenn Sie sich für ein Angebot entscheiden, wird sie sofort die Flüge buchen, das Hotelzimmer reservieren und einen Mietwagen bestellen. Wenn Sie sich anders besinnen, kein Problem: Flüge, Hotelzimmer und Mietwagen werden storniert, und Sie entscheiden sich für ein anderes Angebot. Während Sie Ihre Buchungen tätigen und revidieren, tun dies Hunderte, vielleicht Tausende anderer Kunden in Reisebüros in ganz Deutschland. Wenn Ihr Wunschhotel zu Ihrer Wunschzeit besetzt war, hat vielleicht ein Kunde in Köln Ihnen das letzte Zimmer eine Sekunde zuvor weggeschnappt.

Um Geschäftsgänge wie diese zu unterstützen, bedarf es eines computergestützten Informationssystems. Es verwaltet große Mengen von Daten, die ständig von vielen Stellen aus zu unvorhersehbaren Zeiten geändert werden und doch korrekt bleiben müssen. Kunden sind nicht zufrieden, wenn Flüge überbucht oder Hotelzimmer mehrfach belegt sind. Ein Hotel kann in seiner Existenz gefährdet werden, wenn Zimmer wochenlang leer bleiben, weil der Computer meint, sie seien belegt.

Informationssysteme sind Softwaresysteme. Software ist das System von Programmen, das einen Computzer erst benutzbar macht: auf der internen Maschinenebene erscheinen Programme und Daten als Myriaden von binären Zeichen, sogenannten Bits; Software schafft von dort die Brücke zu menschlich verständlichen Begriffen wie Zahlen, Texten und Bildern sowie Anweisungen, wie mit ihnen zu verfahren ist.

2 Dateisysteme und Datenbanksysteme

Konzentrieren wir uns auf die Daten, die in einem Informationssystem verwaltet werden. Es gibt zwei Arten von Softwaresystemen hierfür: Dateisysteme und Datenbanksysteme.

** Diese Arbeit wurde von der EU unterstützt unter dem Vertrag ESPRITIV WG 22704 ASPIRE. Eine Kurzfassung erscheint in [Eh98].

** Vortrag vor der Plenarversammlung der Braunschweigischen Wissenschaftlichen Gesellschaft

Dateisysteme organisieren Daten in Dateien. Eine Datei ist typischerweise eine Folge von Datensätzen, von denen jeder aus einer festen oder variablen Zahl von Datenfeldern besteht. Jedes Datenfeld beherbergt einen Attributwert. Datensätze beschreiben die relevanten Eigenschaften von Anwendungsobjekten wie z. B. Personen, Projekten, Firmen oder Einträgen in ein Telefonbuch. Die Eigenschaften werden als Attributwerte beschrieben, z. B. das Geburtsdatum einer Person, der Name einer Firma oder die Nummer eines Telefonteilnehmers. Datensätze werden meist durch ein eindeutiges Schlüsselattribut identifiziert, z. B. die Personalausweisnummer einer Person. Die typischen Operationen, die von Dateisystemen unterstützt werden, sind der Zugriff auf Sätze aufgrund von gegebenen Schlüsselwerten sowie das Einfügen, Löschen und Ändern von Sätzen.

In Datenbanksystemen lassen sich Daten auf flexible Weise dateiübergreifend verknüpfen, pflegen und sichern und können vielen Benutzern als gemeinsame Ressource angeboten werden. Solche Systeme machen die Daten von verschiedenen Seiten aus zugänglich und schützen sie zugleich gegen Eingabefehler sowie gegen unberechtigten Zugriff. Die Software sorgt außerdem dafür, daß Benutzertransaktionen quasi-gleichzeitig und ohne gegenseitige Störung abgewickelt werden können, daß Anwenderprogramme bei Erweiterung oder, Umstrukturierung der Datenbank unverändert weiterlaufen können, und vieles mehr.

Besonders einfach sind die Daten in *relationalen* Datenbanken strukturiert, die heute den Stand der Technik darstellen. Die Daten sind als Sammlungen von Tabellen strukturiert, die wegen ihrer Ähnlichkeit mit einem entsprechenden mathematischen Konzept *Relationen* genannt werden. Die Zeilen stellen Objekte dar, die Spalten deren Eigenschaften in der Form von Attributwerten.

Beispiel 1: Die Datenbank einer Firma GENUG (GEsellschaft für Nahrungs- Und Genußmittel) speichert Daten über Kunden und deren Aufträge sowie über Lieferanten und deren Warenangebote. Drei Relationen erfüllen den Zweck:

KUNDE	KName	KAdresse	Kontostand
	Müller	*Braunschweig*	+ 170
	Meier	*Hannover*	+ 97
	Schulze	*Hamburg*	− 17
	...	...	...

AUFTRAG	KName	Ware	Menge
	Meier	*Mehl*	20
	Müller	*Bananen*	70
	...	...	...

LIEFERANT	LName	LAdresse	Ware	Preis
	Rasch	*Hannover*	*Mehl*	17
	Emsig	*Braunschweig*	*Mehl*	15
	...	...		...

Relationale Datenbanksysteme bieten neben einem programmierten auch einen interaktiven Zugang über den Bildschirm. Der verbreitete Standard ist die Datenbanksprache SQL (*Structured Query Language*), sie gestattet die bequeme Formulierung von Anfragen und Änderungen.

Beispiel 2: Kunden, die ihr Konto überzogen haben, findet man aus der obigen Beispieldatenbank mit der SQL-Anfrage

```
select   KName
from     KUNDE
where    Kontostand < 0
```

Anfragen können auch geschachtelt sein. Zum Beispiel ermittelt man die billigsten Lieferanten von Mehl mit der SQL-Anfrage

```
select   Name, Preis
from     LIEFERANT
where    Ware = „Mehl" and Preis ≤ all  (select  Preis
                                         from    LIEFERANT
                                         where   Ware = „Mehl")
```

Anfragen können sich auch über mehrere Relationen erstrecken. Zum Beispiel stellt die folgende Anfrage die Namen und Kontostände aller Kunden zusammen, die Kaffee bestellt haben.

```
select   KUNDE.KName, Kontostand
from     KUNDE, AUFTRAG
where    KUNDE.KName = AUFTRAG.KName   and   Ware = ‚Kaffee'
```

SQL-Anfragen bestimmen das Ergebnis eindeutig, nicht aber die Auswertestrategie. Hiermit möchte ein Anwender auch nicht behelligt werden. Um so schwerer ist die Bürde für das System: es hat lange gedauert, bis Techniken der *Anfrage-Optimierung* so weit entwickelt waren, daß die Antwortzeiten für relationale Anfragen akzeptabel wurden.

Beispiel 3: Um einen Eindruck zu gewinnen, was mit Optimierung zu erreichen ist, betrachten wir zwei verschiedene Auswertestrategien für die obige Anfrage. Dazu nehmen wir an, daß 100 Kunden und 10000 Aufträge vorliegen, darunter 50 Aufträge für Kaffee. Als Maß für den Berechnungsaufwand wählen wir die Anzahl der Zugriffe zu Tupeln. Dies ist grob vereinfacht, gibt jedoch einen qualitativ zutreffenden Eindruck von dem Optimierungspotential.

Die erste Auswertestrategie geht so vor, wie es die letzte Anfrage im Beispiel 2 in blinder Auslegung nahelegt.

1. *Bilde alle kombinierten Kunden-Auftrags-Tupel* (jeder Kunde mit jedem Auftrag verknüpft). Dies sind 100 × 10000 = 1 Million lesende und noch einmal ebensoviele

schreibende Zugriffe, um das Zwischenergebnis zu speichern. Insgesamt also 2 Millionen Zugriffe.

2. *Selektiere alle Tupel mit der Ware ‚Kaffee'.* Hierzu werden alle 1 Million Tupel des Zwischenergebnisses noch einmal gelesen, um die 50 relevanten Tupel auszuwählen. Diese werden zwischengespeichert.
3. *Ausgabe der Antwort.* Hierzu werden die 50 Tupel wieder gelesen, um von jedem die beiden Attribute KName und Kontostand auszugeben.

Insgesamt ergeben sich 3000150 Tupelzugriffe. Man kann es offensichtlich besser machen:

1. *Selektiere alle Aufträge mit der Ware ‚Kaffee'.* Dies sind 10000 lesende und 50 schreibende Zugriffe.
2. *Ermittle zu jedem Kaffee-Auftrag den Kontostand des Kunden und füge ihn zu dem Auftragstupel hinzu.* Hierzu wird – geeignete Datenorganisation mit direktem Zugriff über Kundennamen vorausgesetzt – bei jedem der 50 Tupel der passende Kontostand mit einem Zugriff gefunden. Dies sind weitere 50 Zugriffe.
3. *Ausgabe der Antwort.* Hierzu werden für jedes der 50 Tupel die beiden Attribute KName und Kontostand ausgegeben.

Insgesamt ergeben sich 10100 Tupelzugriffe, wir haben das Finden der Anwort etwa um den Faktor 300 beschleunigt.

Mit geeigneten Zugriffshilfen kann man es noch wesentlich schneller machen: kann man über einen Index gezielt alle Aufträge für Kaffee zugreifen, so sind nur $3 \times 50 = 150$ Tupelzugriffe nötig. Dies ist gegenüber der letzteren Methode abermals eine Beschleunigung um etwa den Faktor 67, gegenüber der ersteren eine Beschleunigung um etwa den Faktor zwanzigtausend!

Neben den Anfragesprachen und der Auswertung und Optimierung von Anfragen gab es in der Datenbank-Technik eine Reihe weiterer ansehnlicher Fortschritte, so z. B. bei der Einrichtung und Verwaltung spezifischer Benutzersichten auf ein und derselben Datenbank, der Durchführung quasi-gleichzeitiger Transaktionen auf gemeinsamen Datenbeständen, der Verteilung einer Datenbank auf mehrere Orte usw.

Und auch bei allen Aspekten des *Entwurfs* von Datenbanken gab es Fortschritte. Um ein mögliches Problem im Beispiel 1 anzudeuten: ist dies Datenbankschema wirklich gut? Oder sollte man die Lieferanten-Relation

LIEFERANT	LName	LAdresse	Ware	Preis
	...	...		...

nicht besser in zwei Relationen aufteilen, eine für die Adressen und eine für die Warenangebote?

LIEFADR	LName	LAdresse
	...	...

ANGEBOT	LName	Ware	Preis
	...	...	...

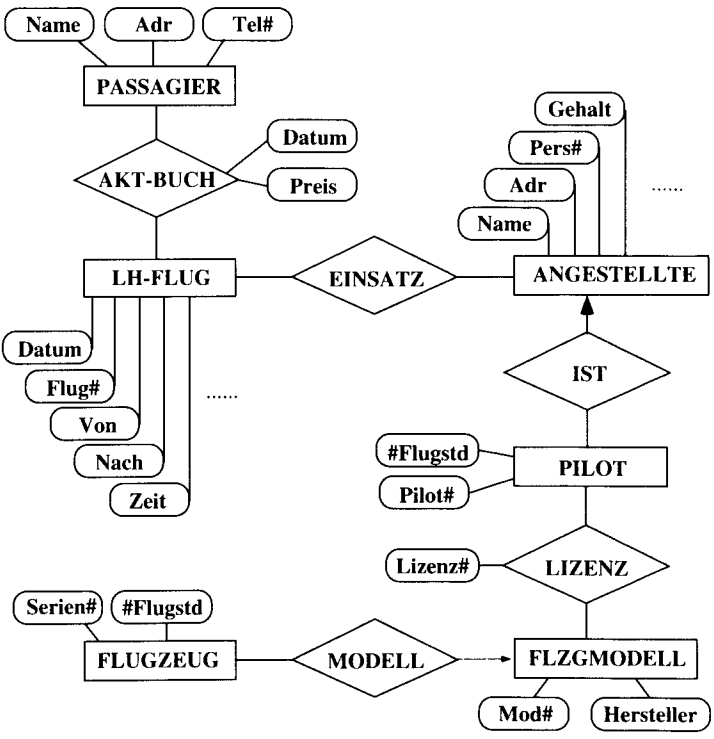
Der Vorteil ist, daß bei vielfältigem Warenangebot eines Lieferanten seine Adresse nur einmal gespeichert ist. Das spart nicht nur Speicherplatz, man vermeidet auch die Gefahr von inkonsistenten Änderungen. Und es sind Zustände problemlos darstellbar wie der, in denen ein Lieferant im Moment keine Waren anbietet, aber noch eine Adresse hat.

Andererseits wären Anfragen nach Angeboten von Lieferanten mit ihren Adressen aufwendiger in der internen Verarbeitung, da auf zwei Tabellen zugegriffen werden muß.

3 Konzeptionelle Modellierung und Spezifikation

Weder Datei- noch Datenbanksysteme bieten in ihren Datenstrukturen direkte Entsprechungen zu Anwendungsobjekten: die Welt besteht nicht aus Datensätzen oder Relationen. Bei der *konzeptionellen Modellierung* von Informationssystemen, die dem Datenbank-Entwurf vorangeht, benötigt man Darstellungsmittel, die die Anwendungsobjekte wie Flüge, Buchungen, Passagiere usw. direkt darzustellen gestatten. Eines der ersten hierfür vorgeschlagenen Datenmodelle war das *Entity-Relationship-Modell*. Entities repräsentieren Anwendungsobjekte mit ihren Attributen, und Relationships repräsentieren Beziehungen zwischen Entities. ER-Modelle lassen sich graphisch darstellen: Rechteckboxen stellen Entities dar, Rundboxen Attribute und Rautenboxen Beziehungen.

Beispiel 4:



Pfeilköpfe an Beziehungen bedeuten Eindeutigkeit: jeder Pilot ist zugleich ein Angestellter, und zwar nur einer, und jedes Flugzeug ist Exemplar eines eindeutigen Flugzeug-Modells wie z. B. Airbus A320 oder Boeing 737.

Das ER-Modell findet in der einen oder anderen Variante oder Erweiterung zunehmend Verbreitung für die praktische konzeptionelle Datenmodellierung, jedoch gibt es auch andere Ansätze. Und es gibt Verfahren, wie man aus solchen konzeptionellen Modellen relationale Datenbankschemata gewinnen kann, die dieselbe Information darstellen.

Ein Informationssystem besteht nun nicht nur aus Daten, sondern auch aus Operationen zur Verarbeitung der Daten. Auch hier gilt es, zur Vorbereitung des Entwurfs eine Darstellung auf angemessener Abstraktionsebene zu finden. Statt fertiger Programme mit detaillierten Ausführungsbeschreibungen sind hier möglichst knappe Wirkungsbeschreibungen gefragt. Um z. B. eine Operation sqrt(x) zur Berechnung der Quadratwurzel von x zu spezifizieren, reicht es, für das Ergebnis die Eigenschaft $\text{sqrt}(x) \times \text{sqrt}(x) = x$ zu fordern. Ein Berechnungsverfahren ergibt sich daraus noch nicht, dies ist Sache der späteren Ausführung durch Programmierer.

In der Tat sind Programmierer die ausführenden Organe, gleichsam die Maurer der Software-Herstellung. Wir brauchen sie, aber was wir auch sehr dringend brauchen sind Software-Ingenieure und Architekten, die deren Arbeit vorbereiten und organisieren und dafür sorgen, daß am Ende alles zusammenpaßt. Diese Aufgabe erfordert Modellierung und Spezifikation mit der gebotenen Verständlichkeit und Verbindlichkeit, und dies erfordert Abstraktion und Präzision – traditionelle Domänen der Mathematik und der Logik. So werden denn auch *formale Methoden*, d. h. der Gebrauch von Konzepten aus Mathematik und Logik für die Software-Technik, breit erforscht und diskutiert. In der Praxis sind formale Methoden bislang kaum verbreitet, jedoch gibt es aus größeren Fallstudien und Pilotprojekten ermutigende Erfahrungen.

4 Objektmodellierung

Die Behandlung von Daten und Verarbeitungsprozessen sind traditionell verschiedene Gebiete der Informatik, mit verschiedenen Begriffen, Theorien, Techniken und Herstellern. Daher rührt u. a. die notorische Schwachstelle in der Kooperation zwischen Anwenderprogrammen und Datenbanken, der sogenannte *impedance mismatch*. Beim konventionellen und heute noch meist praktizierten Software-Entwurf wird dies ebenfalls deutlich: die Daten und Operationen werden zu Beginn getrennt und verschiedenen Teams zur Bearbeitung übergeben. Am Ende paßt es dann oft nicht recht zusammen.

Zur Modellierung von Informationssystemen gilt es zunächst, Daten und Operationen besser miteinander zu verbinden. Hier ist nun das Software-Konstrukt des *Objekts* hilfreich, eine einleuchtende und durchaus nicht neue Idee, die Schübe von Forschungen und Entwicklungen hervorgebracht hat: zu objekt-orientierten Programmiersprachen, objektorientierten Datenbanksystemen und ebensolchen Methoden des Software-Entwurfs.

Ein Objekt stellt eine eigene autonome Speicher- und Verarbeitungseinheit dar. Es kapselt Daten und Operationen ein und macht sie von außen über eine wohldefinierte Schnittstelle zugänglich. Mit diesem Konzept ist es möglich, Anwendungsobjekte vollständig darzustellen, d. h. mit ihrer statischen Struktur und ihrem dynamischen Verhalten.

Auf der Grundlage dieser Ideen begann vor etwa zehn Jahren ein spezifisches Forschungsprogramm zur konzeptionellen Modellierung und Spezifikation von Informationssystemen [SSE87]. Der Ansatz kombiniert und vereinheitlicht die Behandlung von Daten und Operationen, plädiert für den Gebrauch temporaler Logik und legt großen Wert auf das Verständnis der theoretischen Grundlagen. Ein System wird als Gemeinschaft von Objekten angesehen, die synchron und symmetrisch durch gemeinsame Ereignisse miteinander kommunizieren. Systemverhalten wird konsequent aus der Sicht der Objekte beschrieben.

Diese Forschungsarbeiten wurden im Rahmen mehrerer nationaler und europäischer Projekte gefördert, begleitet von Industriekooperationen und Fallstudien. An der TU Braunschweig wurden die Sprachen TROLL und OMTROLL entwickelt [DH97], eine textuelle und eine graphische Sprache zur Spezifikation von Objektsystemen. Darüber hinaus wurde eine Methode entwickelt, die einen Leitfaden zur Objektmodellierung anhand dieser Sprachen bietet. Eine Entwicklungsumgebung mit Software-Werkzeugen zur Unterstützung des Vorgehens befindet sich in der Entwicklung. Die Arbeiten [EH96, HDK+97] geben einen Einblick in den Stand der Forschung.

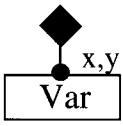
Der TROLL-Ansatz favorisiert vier Modellsichten: ein *System-Modell*, in dem die Gesamtarchitektur des Systems beschrieben wird, ein *Objekt-Modell* zur detaillierteren Beschreibung der Objektklassen des Systems, ein *dynamisches Modell* zur Beschreibung des dynamischen Verhaltens der Objekte jeder Objektklasse, und ein *Kommunikations-Modell* zur Beschreibung der Objekt-Kommunikation und damit der globalen Prozesse.

Die Methode schlägt vor, wie nacheinander Objekte und ihre Beziehungen in der Anwendung zu identifizieren und wie dann die Schnittstellen der Objekte und ihr internes Verhalten zu spezifizieren sind. Die graphische Sprache OMTROLL erlaubt die übersichtliche graphische Darstellung des Systems und seiner Teile, kann jedoch nicht alle Verhaltensregeln beschreiben. Erst die textuelle Darstellung in TROLL erlaubt die vollständige Spezifikation aller Aspekte.

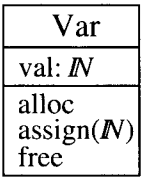
Werkzeuge sollen den Umgang mit Spezifikations-Dokumenten erleichtern und die Prüfung und Erprobung des Entwurfs in möglichst frühem Stadium ermöglichen. Dazu gehören Editoren zur Erstellung der OMTROLL- und TROLL-Spezifikationen, Syntax-Checker für diese Editoren, ein Animator zum Erproben der Funktionsweise der spezifizierten Objekte und ihrer Kommunikation, Werkzeuge zur Analyse der Spezifikation (Model Checker, Beweissystem), Werkzeuge zum Übersetzen der Spezifikation in ausführbare Programme sowie Werkzeuge zum Erzeugen von Testdaten und Testfällen, mit denen die korrekte Funktionsweise des implementierten Systems überprüft werden kann.

Beispiel 5: Um einen Eindruck von OMTROLL und TROLL zu geben, betrachten wir ein sehr einfaches Informationssystem. Es besteht aus zwei kommunizierenden Zustandsvariablen x und y. Eine Zustandsvariable kann alloziert, d.h. durch Bereitstellen von Speicherplatz geschaffen werden (*alloc*). Ihr können dann immer wieder neue Werte zugewiesen werden (*assign(n)*), bis sie freigegeben wird (*free*). Eine Variable speichert immer den zuletzt zugewiesenen Wert (Attribut *val*). Der Einfachheit halber seien alle Werte natürliche Zahlen, also Elemente von $\mathbb{N}$. Als Beispiel für eine simple Kommunikation werde y jedesmal auf Null gesetzt, wenn x einen runden, also einen durch 10 teilbaren, Wert bekommt: bei *x.assign(3)* (d.h. Aufruf der Operation *assign(3)* im Objekt x passiert also nichts, bei *x.assign(30)* wird aber zugleich *y.assign(0)* aufgerufen.

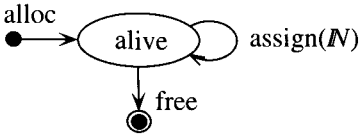
Das System-Modell zeigt einen Überblick über die Objektpopulation. In komplexeren Beispielen treten hier mehrere Objektklassen auf, ggf. mit Aggregierungs- und Spezialisierungs-Beziehungen zwischen ihnen. Das Innere der Objektklassen ist verborgen, es wird getrennt im Objekt-Modell dargestellt. Das dynamische Modell zeigt die klassenspezifischen Objekt-Verhaltensmuster in der Form von Zustandsautomaten. Das Kommunikations-Modell zeigt die beabsichtigte Kommunikation zwischen den Objekten in der Form von Aktionsaufrufen. Ein Aufruf kann an Bedingungen geknüpft werden, und mit Hilfe von Parametern können Daten übertragen werden.



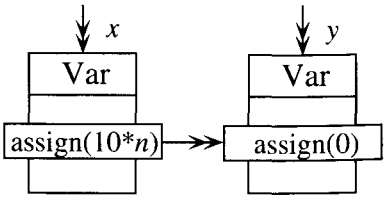
System-Modell



Objekt-Modell



dynamisches Modell



Kommunikations-Modell

Die textuelle TROLL-Darstellung dieses Beispiels könnte folgendermaßen aussehen (wobei in den Axiomen eine temporale Logik benutzt wird, die im Kern der gegenwärtigen Version TROLL3 nicht unterstützt wird).

Hier ist zunächst die Spezifikation der Objektklasse `Var`.

```
object class Var
  attributes val:nat;
  actions * alloc; assign(value:nat); +free;
  behavior assign(n) do val:=n od;
  axioms
    occurred alloc ⇒
      (not enabled alloc and enabled assign(n) and enabled free)
        until occurred free,
    occurred free ⇒
      (not (enabled alloc or enabled assign(n) or enabled free)
end;
```

Die System-Spezifikation sieht dann folgendermaßen aus:

```
object system Kommunizierende_Variable
  object class Var
    ... (s.o.)
  end;
  objects x,y:Var;
  behavior x.assign(10*n) do y.assign(0) od
end
```

5 Formale Semantik

Die semantische Grundlage von TROLL ist eine lineare temporale Logik mit Kommunikation, genannt D_0. Untersucht wird auch eine erweiterte Logik D_1, aber darauf soll hier nicht eingegangen werden.

Es sei eine Menge $Id = \{1, \ldots, n\}$ von *Objekt-Identitäten* gegeben. Für jedes Objekt $i \in Id$ sei eine Menge P_i von *Zustands-Prädikaten* gegeben, und darin sei eine Teilmenge $C_i \subseteq P_i$ von *Kommunikations-Prädikaten* ausgezeichnet. *false* bezeichne den logischen Wert „falsch" bzw. den logischen Widerspruch, und $\Rightarrow$ bezeichne die logische Implikation „wenn ... dann ...".

Die Syntax von D_0 besteht aus lokalen Logiken für jedes Objekt,

$$D_0 ::= i.D_0^1 \mid \ldots \mid D_0^n \ .$$

Die Notation bedeutet, daß jede Formel aus D_0 entweder aus D_0^1 oder ... oder aus D_0^n ist. Die i-te lokale Logik ist syntaktisch beschrieben durch

$$\begin{aligned} D_0^i &::= H_0^i \mid i.CC_0^1 \\ H_0^i &::= P_i \mid \text{false} \mid (H_0^i \Rightarrow H_0^i) \mid (H_0^i \,\mathfrak{U}\, H_0^i) \mid (H_0^i \,\mathfrak{S}\, H_0^i) \\ CC_0^i &::= (C_i \Rightarrow 1.C_1) \mid \ldots \mid (C_i \Rightarrow n.C_n) \end{aligned}$$

Jede lokale Formel für das Objekt i beginnt mit der Markierung $i.$ gefolgt von einer internen Formel (H_0^i) oder einer Kommunikationsformel (CC_0^i). H_0^i ist eine lineare temporale Logik mit temporalen Operatoren $\mathcal{U}$ für *until* und $\mathcal{S}$ für *since*: $\varphi\ \mathcal{U}\ \psi$ bedeutet, daß φ vom nächsten Zustand an immer gilt, bis das nächstemal ψ gilt; $\varphi\ \mathcal{S}\ \psi$ bedeutet, daß φ bis zum vorigen Zustand immer gegolten hat, seit das letztemal ψ gegolten hat.

Beispiel 6: Die Zustandsvariablen x und y werden in ihrem Verhalten durch die folgenden Axiome beschrieben. Dabei benutzen wir ein Aktionssymbol a auch als Prädikatensymbol: es drückt aus, daß die Aktion a beim Eintritt in den aktuellen Zustand ausgeführt wurde. Die Notation $\triangleright a$ bedeutet, daß die Aktion a *enabled* ist, d.h. daß sie als nächstes ausgeführt werden *kann*. Bei einem durch Aktion a ausgelösten Zustandsübergang muß vorher also $\triangleright a$ gelten, und nachher gilt a. In den folgenden Axiomen benutzten wir einen von $\mathcal{U}$ abgeleiteten temporalen Operator $\varphi\ \mathcal{U}^\circ\ \psi$. Er bedeutet, daß φ vom *jetzigen* Zustand an immer gilt, bis das nächstemal ψ gilt.

Aufgrund der Klassenspezifikation im Beispiel 5 ergeben sich lokal für x folgende Regeln. $\wedge$ bezeichne das logische „und", es läßt sich wie alle anderen logischen Verknüpfungen mittels der gegebenen *false* und $\Rightarrow$ definieren.

$x.\ \text{alloc} \Rightarrow (\neg\triangleright\text{alloc} \wedge \triangleright\text{assign}(u) \wedge \triangleright\ \text{free})\ \mathcal{U}^\circ\ \text{free},$
$x.\ \text{free} \Rightarrow \neg\triangleright\text{alloc} \wedge \neg\triangleright\text{assign}(u) \wedge \neg\triangleright\text{free},$
$x.\ \text{assign}(u) \Rightarrow \text{val} = v\ \mathcal{U}^\circ\ (\text{assign}(w) \vee \text{free})$

Für y gelten die gleichen Regeln, beginnend mit $y.$ anstelle von $x.$

Aufgrund der Systemspezifikation im Beispiel 5 ergibt sich für x darüber hinaus die Kommunikationsregel

$$x.\ \text{assign}(10 \times n) \Rightarrow y.\ \text{assign}(0)\ .$$

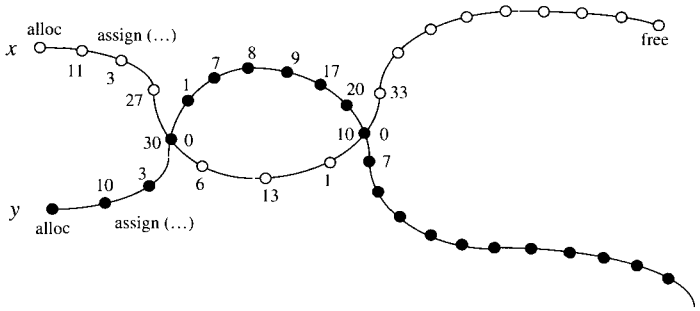
Solche Regeln lassen sich interpretieren im mathematischen Modell der System-Lebensläufe, bestehend aus je einem Lebenslauf für jedes Objekt im System. Ein Lebenslauf ist eine endliche oder unendliche Folge von Ereignissen. Jedes Ereignis ist behaftet mit einer Menge von Zustands-Prädikaten, also Feststellungen über den Zustand des Objekts bei diesem Ereignis. Dazu gehört die Aussage darüber, welche Aktionen gerade stattgefunden haben.

Ein Kommunikationsereignis erscheint in dem Modell als ein gemeinsames Ereignis in verschiedenen Objekt-Lebensläufen. Die Zustände der an einem Kommunikationsereignis teilhabenden Objekte sind natürlich im allgemeinen verschieden.

Beispiel 7: Der Variablen x werden nach der Allozierung der Reihe nach die Werte 11, 3, 27, [30], 6, 13, 1, [10], 33, ... zugewiesen, bevor sie einige Schritte später freigegeben wird. Die Zuweisung von 30 und 10 erfordert jeweils Kommunikation mit der Variablen y, angedeutet durch die eckigen Klammern: y wird dort jeweils auf 0 gesetzt. Zwischendurch kann y mit eigenen Angelegenheiten beschäftigt sein. Eine mögliche Abfol-

ge von Wertzuweisungen an y ist 10, 3, [0], 1, 7, 8, 9, 17, 20, [0], 7, ... Der Lebenslauf von y kann irgendwann enden, muß aber nicht.

Das folgende Diagramm veranschaulicht diese Lebensläufe von x und y.



Dies ist nur ein möglicher vernetzter Lebenslauf, viele andere sind möglich. Die volle Bandbreite des Verhaltens wird erst durch die Gesamtheit aller möglichen vernetzten Lebensläufe charakterisiert. Dies Verhaltensmodell ist eine Ereignisstruktur nach Winskel.

Die Semantik einer D_0 Spezifikation ist nun die Menge aller vernetzten Lebensläufe, die die Axiome erfüllen. Diese Festlegung spiegelt das Prinzip maximaler Liberalität wider: alle Verhaltensweisen sind erlaubt, sofern ihnen nicht ausdrücklich Regeln entgegenstehen.

Als formale Semantik einer TROLL-Spezifikation wird die der ihr zugrundeliegenden D_0-Axiome festgelegt. Auf diese Weise gelingt es, jeden syntaktisch gültigen TROLL-Text zu „verstehen": er beschreibt eine Menge von D_0-Formeln, die ein genaues mathematisches Modell für das erlaubte Verhalten des Systems definieren. Dies ist eine notwendige Voraussetzung dafür, daß man präzise entscheiden kann, ob ein implementiertes System der Spezifikation genügt oder nicht.

6 Schlußbemerkungen

Das Ziel der hier dargestellten Forschungsarbeiten ist es, zur Entwicklung von Techniken für die Konstruktion solider und verläßlicher Informationssysteme mit zugesicherten Eigenschaften sowie deren theoretische Absicherung beizutragen. Weitere Untersuchungen auf allen angesprochenen Gebieten sind nötig, von den theoretischen Grundlagen über Sprachen, Methoden und Werkzeuge bis hin zu Fallstudien und praktischen Pilotprojekten.

Ein besonderes Augenmerk wird dabei auf die *Integration* von Methoden zu richten sein, die einzeln bereits gut verstanden werden, die Erfordernisse der Praxis aber nicht ausreichend abdecken.

Zu einer Verbesserung der Praxis gehört auch, daß erprobte neue Techniken denen nahegebracht werden, die sie anwenden sollen. Neben Anstrengungen in Lehre und Fortbildung sind hier vor allem Erfolgsmeldungen aus der Praxis dienlich, die die Vorteile neuer Methoden vor Augen führen und die Erkenntnis vermitteln, daß Innovation nötig ist, um wettbewerbsfähig zu bleiben. Über solche Fälle wird zunehmend berichtet.

Die Entwicklung der Software-Technik im allgemeinen und des Entwurfs von Informationssystemen im besonderen hat indes nicht nur technische Aspekte. Um die sich abzeichnenden Möglichkeiten zielgerichtet und ökonomisch zu nutzen, ist ein umfassendes Qualitätsmanagement für Software fortzuentwickeln, bestehend aus ständiger Qualitätsplanung, -kontrolle und -verbesserung.

Literaturhinweise

Die Hinweise beschränken sich auf die in diesem Beitrag angesprochenen eigenen Forschungsarbeiten. Natürlich haben viele andere Autoren Grundlagen, Ideen und Anregungen beigesteuert. Eine einigermaßen vollständige Darstellung dieser Einflüsse und Kooperationen würde den hier gesteckten Rahmen sprengen. In den nachfolgend zitierten Arbeiten finden sich auch hierzu detaillierte Hinweise.

[DH97] G. Denker and P. Hartel. TROLL – An Object Oriented Formal Method for Distributed Information System Design: Syntax and Pragmatics. *Informatik-Berichte 97-03, TU Braunschweig* 1997.

[EH96] H.-D. Ehrich and P. Hartel. Temporal specification of information systems. In A. Pnueli and H. Lin, editors, *Proc. Int. Workshop in Honor of Chih-Sung Tang, World Scientific, Singapore*, pages 43–70, 1996.

[Eh98] H.-D. Ehrich. Konzeptionelle Modellierung von Informationssystemen (Kurzfassung). Erscheint in Carolo-Wilhelmina Mitteilungen, Heft I/1998.

[HDK+97] P. Hartel, G. Denker, M. Kowsari, M. Krone, and H.-D. Ehrich. Information systems modelling with TROLL formal methods at work. *Information Systems*, 22(23): 79–99, 1997.

[SSE87] A. Sernadas, C. Sernadas, and H.-D. Ehrich. Object-oriented specification of databases: An algebraic approach. P. Hammerslay, editor, *Proc. 13th Int. Conf. on Very Large Databases, VLDB '87, pages 107–116, Brighton, 1987. Morgan–Kaufmann, Palo Alto, 1987.*

Prof. Dr. rer. nat. Hans-Dieter Ehrich
Abteilung Datenbanken, Technische Universität Braunschweig,
Postfach 3329 · D-38023 Braunschweig, Germany
http://www.cs.tu-bs.de/idb/welcome_e.html
HD.Ehrich@tu-bs.de